代码说明

一、 需求背景

需求来源：产品经理。

需求背景：客户端在会中时，用户变更时，能够实时更新用户的状态，包括退出，进入，信息，状态变更等等。在会中的各个界面可以实时收到用户状态变更消息和对应数据，从而控制 UI 进行相应显示。其中就包含参会人界面。当时 Android 原来的设计实现存在问题，不能很好的满足需要，正好在 3.11 版本中进行会中的整体重构。

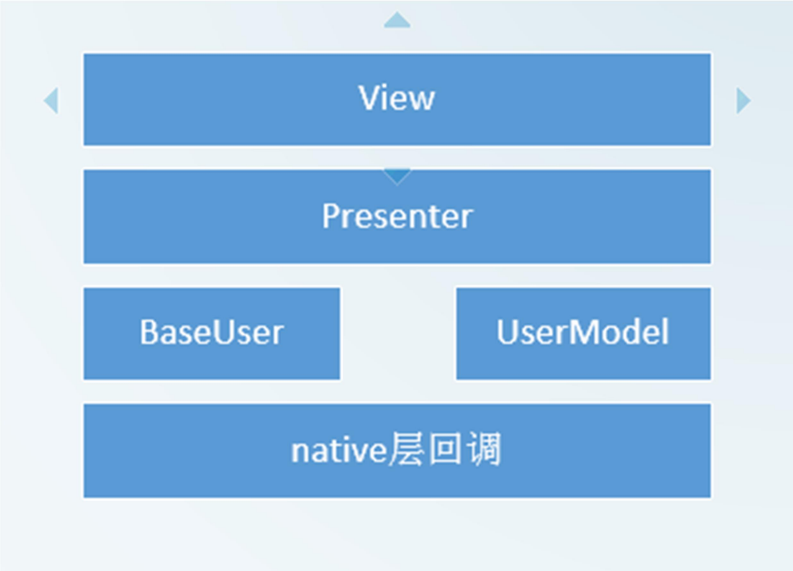
需要解决的问题：

1、 如何实现对用户的统一管理
2、 如何将用户数据细节隐藏起来，封装成接口方法。
3、 如何实现用户状态的变更可以同时通知到多个观察者，由各上层业务模块只关注和处理自己需要的数据
4、 上层业务层和 View 如何实现更高程度的分离。
5、 大量用户同一时间段用户进入或状态变更时，如何减少对参会人界面 UI 的重复绘制。

二、 设计思路

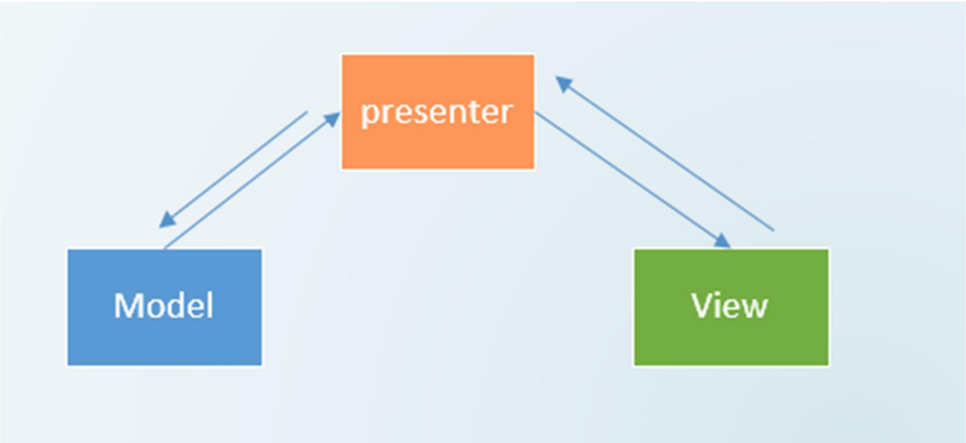
设计采用分层设计，总体来说是:

1、 native 层回调 Java 层通知到 model（Android 与 TV 通用业务层），封装 RoomUserInfo 为 BaseUser，对外界隐藏数据细节。
2、 mode 层分发用户信息到 Android 上层业务层(presenter)
3、 由上层业务控制 UI 变更。



注：BaseUser 和 UserModel 属于基础通用类，其给所有需要会中用户信息的模块提供服务。本次中使用的参会人使用只是其中之一。

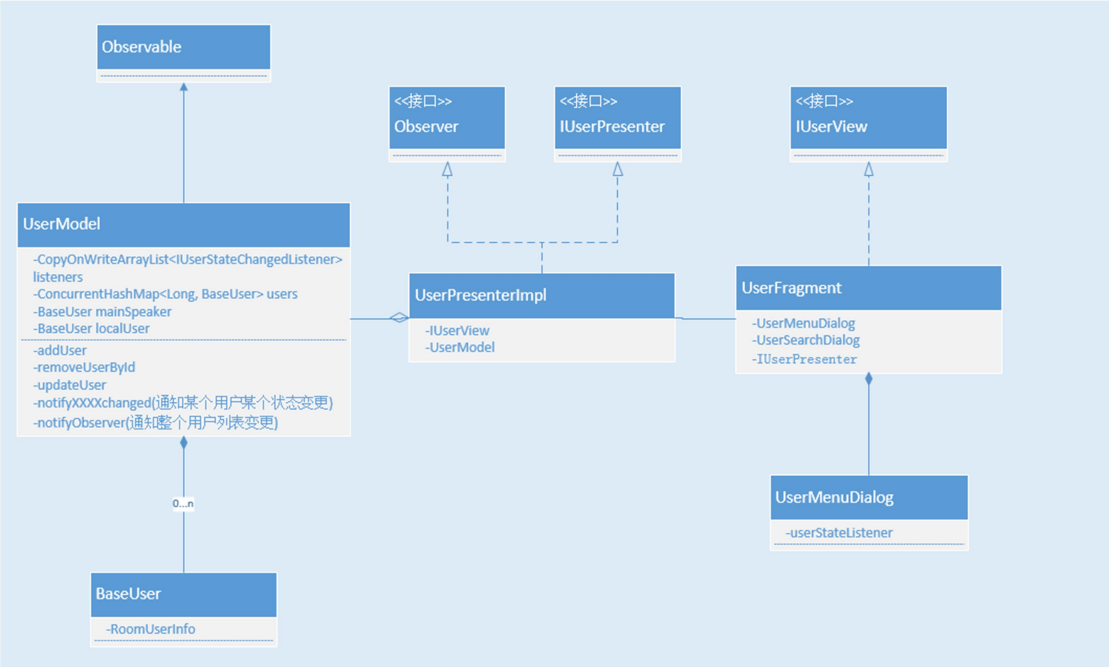
上层业务与 UI 之间采用更适合 Android 的 MVP 设计：



注：MVP 结构是从 MVC 结构演化而来，其与 MVC 最大的不同的是 View 不再直接和 Model 通信，转而都通过 Presenter 来进行，所有的交互都发生在 Presenter 内部，View 和 Presenter 之间是通过接口来通信的，从而达到 View 与业务更高程度的分离，方便后续修改与维护，测试。

此种设计也是目前 Android 项目使用的一种通用的从 Model 层到上层业务再到 UI 的设计方案。

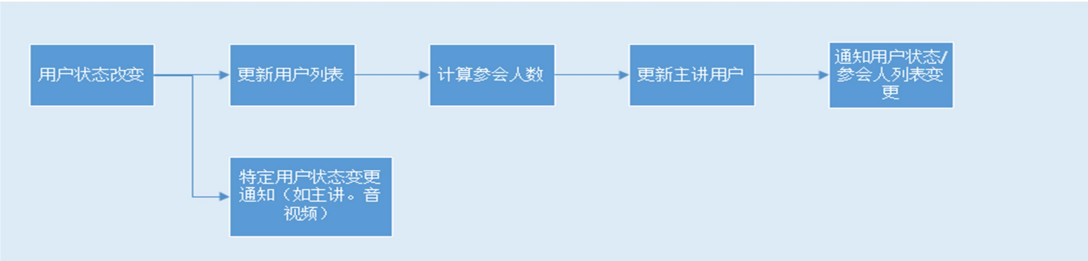
三、实现说明



1、 在会中用户变化时，MeetingRoomNotify 调用对用的 UserModel 方法（add, update, remove，notifyXXXChanged）对用户进行操作

2、 在 UserModel（add, update, remove）方法被调用时，将底层发送的 RoomUserInfo 根据用户角色封装成对应的 User,添加/更新/删除到用户列表

3、 更新完用户列表后，通过计算参会人数等一系列操作根据不同的状态变更调用不同接口通知其观察者。



4、 在参会人列表的 Presenter 收到用户状态变更时，根据当前显示的界面刷新不同的用户列表

5、 UserFragment 和 UserPresenterImpl 通过接口进行交互，UserFragment 作为 View 将所有业务逻辑转交给 Presenter 处理，处理完后，反过来控制 UI 展示。（如 Item 点击事件，搜索和弹出菜单显示等等。）

四、代码文件

类	重点检视	功能与用途
BaseUser.java	是	1、封装底层返回的 RoomUserInfo，隔离具体的数据，提供接口方法给外界调用。 2、本身不具有复杂的逻辑，主要基于 RoomUserInfo 的权限检查和状态判断。
UserModel.java	是	1、管理会中用户与会议室参会人数。 2、管理会中主讲用户和主讲相关操作。 3、统一管理各上层模块对会中用户状态的监听，广播会中用户状态变更，提供便捷接口，各模块只实现自己关心的状态变更的回调接口。
IUserFragmentContract.java	否	1、参会人列表 View 和 Presenter 的接口契约类，定义了 UserFragment 和 UserPresenterImpl 之间进行交互的接口。
UserFragment.java	是	1、参会人界面的 UI 类，包括用户列表的显示，滑动的操作，显示搜索参会人界面，用户弹出菜单等等。
UserPresenterImpl.java	是	参会人界面的上层业务类: 1、控制参会人界面的 UI 显示。 2、控制搜索参会人界面。 3、监听会议室用户变更。 4、控制搜索参会人界面、弹出菜单界面的显示。

UserMenuDialog.java	是	用户弹出菜单主要关注： 1、支持显示时，实时更新用户昵称和授权主讲权限。（UserStateChangedListener）

注：UserFragment 和 UserPresenterImpl 此处主要是想体现使用 MVP 设计时，业务与 UI 的分离以及业务控制 UI 变化的思想，其具体业务逻辑可不关注。同样 UserMenuDialog 是想体现即使是一个临时的界面对象也可以很方便的监听用户状态变化。

五、总结

优点：

1、 封装 RoomUserInfo 成为 BaseUser，将具体的数据隐藏起来，提供接口给外界调用，让调用者从原来的通过具体数据判断中解耦出来。

2、 会中用户统一管理，并通过观察者模式在用户状态发生变更时进行通知，从而由原来的不同模块在一个回调中处理变为各自模块自行监听处理，减少模块之间的耦合。

3、 通过提接口的便捷实现类 SimpleUserStateChangedListener，让各模块可以只重写自己需要监听的接口，减少重复代码。同时采用此种设计在新增回调接口方法时，无须对原有代码进行任何修改。

4、 应用上层采用 MVP 架构设计，将业务和 view 之间通过接口隔离后进行更高程度的解耦，提高了代码的可维护性，可扩展性和可测试性。

缺点：

1、 在 UserModel 中是否可以将 IUserStateChangedListener 接口直接使用其便捷类 SimpleUserStateChangedListener 代替，因为在目前的业务中和后续业务中，不可能出现一个模块同时需要监听所有状态变化的情况，如果有，那就要考虑模块设计是否合理了。可能的改进措施：删除接口，直接使用简单便捷类 SimpleUserStateChangedListener 代替。

2、 在 IUserStateChangedListener 接口的 onUserChanged 方法功能定位不是非常清晰，包含太广，与其他几个方法在功能上存在一定的重复。改进措施：重新梳理该接口方法的功能，移除其与已经有的接口的重复功能（不再被回调）。

3、 IUserStateChangedListener 回调接口类以及其便捷类是否可以移动到一个单独的类文件中，减小 UserModel 的大小和提高其内聚性。